

UNIVERSIDAD DE CONCEPCIÓN - CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INDUSTRIAL

*Minimizing makespan on identical parallel machines
with one preemption and batch completion constraints.*

por
Ignacio A. Sepúlveda Medina

Profesor Guía:
Dr. Carlos Herrera L.

Concepción, enero de 2021



Tesis presentada a la

DIRECCIÓN DE POSTGRADO
UNIVERSIDAD DE CONCEPCIÓN



Para optar al grado de

MAGÍSTER EN INGENIERÍA INDUSTRIAL

RESUMEN

Minimizing makespan on identical parallel machines with one preemption and batch completion constraints.

Ignacio A. Sepúlveda Medina

Concepción, Enero de 2021

PROFESOR GUIA: Dr. Carlos Herrera L.

PROGRAMA: Magíster en Ingeniería Industrial

En la siguiente investigación se presenta un problema de asignación de máquinas paralelas sin interrupción de procesamiento de los trabajos y considerando restricciones para el completamiento de los lotes que obligan a procesar cierta porción de cada uno de ellos ($h\%$) durante un período de tiempo determinado. El objetivo es minimizar una función de costo total que está compuesta por el costo del *makespan* y por costos de *setup* dependientes de la secuencia. Este problema tiene su origen en el área de la salud, específicamente, en un programa de salud móvil.

Se propone un modelo de programación lineal entera mixta (MILP) para resolver el problema de manera exacta. Como el problema es NP-Hard, se propone un algoritmo genético cuya estructura se basa en el algoritmo propuesto en Prins (2004), donde las características que más destacan son una heurística para la división de trabajos entre las máquinas y el proceso de mutación aplicado basado en búsqueda local. A partir de ello, se obtienen dos variantes del algoritmo. El desempeño del algoritmo se compara con el modelo propuesto y, además, con otro algoritmo genético propuesto en la literatura. La eficiencia de los algoritmos desarrollados es validada con la calidad de los resultados basados, principalmente, en los valores de GAP obtenidos.

Palabras Claves: Máquinas paralelas, Función de costo total, Costos de setup, Programación lineal entera mixta, Algoritmo genético, Lotes.

ABSTRACT

Minimizing makespan on identical parallel machines with one preemption and batch completion constraints.

Ignacio A. Sepúlveda Medina

Concepción, January 2021

THESIS SUPERVISOR: Dr. Carlos Herrera L.

PROGRAM: Master in Industrial Engineering

We consider the parallel machine scheduling problem without preemption adding a batch completion constraint that forces to process a certain percentage $h\%$ of each batch during a period of time. The objective is to minimize the total cost that is compounded by a makespan cost and by the sequence dependent setup cost of the schedule. This problem comes from a real logistics situation that involves the application of a mobile health program.

A MILP model to solve the problem exactly is proposed. As the problem is NP-Hard, a genetic algorithm structure based on the algorithm proposed in Prins (2004) is developed, where the main features are a *Splitting job heuristic* and the mutation process. From it, we obtain two variants of the algorithm. The algorithm performance is compared with the MILP model and with an adapted version of another algorithm from the literature. The efficiency of the proposed algorithms is validated by the quality of the results, based on the GAP of them mainly.

Keywords: Scheduling, Total cost, setup cost, Mixed integer linear programming, Genetic algorithm, Batch constraints.

Contents

1	Introduction	1
1.1	Problem description	1
1.2	Contributions	2
2	Literature review	5
3	Methodology	11
3.1	Problem complexity	11
3.2	Mathematical formulation	11
3.3	A genetic algorithm for the $Pm d_{batch}, h\%_{batch} TC$	15
3.3.1	Solution representation and evaluation	16
3.3.2	Crossover	18
3.3.3	Mutation operator	19
3.3.4	Population structure and initialization	22
3.3.5	Iteration description	22
3.3.6	Main phase and restarts	23
4	Computational results	24
4.1	Benchmark instances	24
4.2	Implementation	24
4.3	An additional comparison	25
4.4	Parameters setting	25
4.5	Experiments on Algorithms	27
4.6	Effect of the movements in the local search	28
4.7	Comparison with the mathematical model	31
4.8	Comparing with the literature	31
4.8.1	Effect of T'	32
5	Conclusions and future works	36

List of Figures

1	Scheduled obtained considering batch due date.	3
2	Scheduled obtained without considering batch due date.	3
3	Chromosome sample	16
4	Splitting procedure sample.	18
5	Example of OX crossover	19
6	M_5 move sample	20
7	M_8 move sample	20
8	Number of successful insertions vs. Number of iterations in instance 6 of GA_δ	30
9	Number of successful insertions vs. Number of iterations in instance 10 of GA_δ	30
10	Number of successful insertions vs. Number of iterations in instance 21 of GA_∞	30
11	Number of successful insertions vs. Number of iterations in instance 21 of GA_δ	30
12	Number of successful insertions vs. Number of iterations in instance 24 of GA_∞	31
13	Number of successful insertions vs. Number of iterations in instance 24 of GA_δ	31
14	Box plot of the GAP obtained in instances 1-5	32
15	Box plot of the GAP obtained in instances 1-10	32
16	Optimum solution for instance 17 with the original T' value	33
17	Optimum solution for instance 17 with a 25% of T' increase	33
18	Optimum solution for instance 17 with a 50% of T' increase	34

List of Tables

1	Jobs data	2
2	Setup costs	2
3	Summary of literature review.	10
4	General description of instances set 1 and 2.	24
5	General description of instances set 3	25
6	Parameters setting for NSGA	26
7	Parameters setting for the GA	26
8	Parameters setting in restart phases for GA	26
9	Preliminary testing of algorithms	27
10	Results on set 1 and 2	28
11	Results on set 3	29
12	Model and algorithm sensitivity respect to T'	35



1. Introduction

1.1. Problem description

The non-preemptive parallel machine scheduling problem is an NP-Hard combinatorial optimization problem where a set of jobs is to be scheduled in a set of homogeneous machines without interrupting (preemption). This problem has been intensively studied in the literature and have broad applicability in industry. In this work, we study an extension of the problem mentioned above, including batch due dates to minimize sequence-dependent setup and makespan-related costs, arising in a real-life application in healthcare management.

The problem addressed in this study can be stated as follows. A set of n jobs should be scheduled on a set of m identical parallel machines without preemption. Each job is available at time 0 and has a processing time P_j (integer number), $j = 1, 2, \dots, n$. Each job belongs to a batch $k \in B$, $B = 1, 2, \dots, b$, each having a common due date T' at which $h\%$ of the total processing time of the batch k must be processed. Additionally, there is a sequence-dependent a setup cost, c_{ij} , $i, j = 1, 2, \dots, n$, incurred when job j is scheduled after job i , and a fixed machine setup-time s after processing each job. The aim of the problem is to find a schedule that minimizes the sequence-dependent and makespan related cost. Using the three field notation proposed by Graham et al. (1979), we denote the problem by $P_m|d_{batch}, h\%_{batch}|TC$. P_m indicates parallel machine, d_{batch} denotes common batch due date, $h\%_{batch}$ indicates that $h\%$ of the total processing time of batch b must be processed before the due date, and TC is equal to $\sum_i c_{ij} + GC_{max}m$, where G represent a cost per unit of time of the schedule.

$P_m|d_{batch}, h\%_{batch}|TC$ arises when planning a health care mobile dental care program. A set of identical dental clinics (machines) must treat low-income students needing dental care in urban and rural schools (jobs). Schools belong to districts (batches) each having a common due date T' . Before the due date, $h\%$ of the total number of patients of each district must be attended, ensuring fairness between the students needing dental care at different districts.

To help the reader to understand the problem, we present an example below with six jobs, two machines, and three batches. Batches 1, 2, and 3 contains jobs $\{1, 3\}$, $\{2, 4\}$, and $\{5, 6\}$ with a total processing time of 168, 172, and 198, respectively. Table 1 contains, for each job, P_i and its assigned batch. We insert two dummy jobs, 0 and 7, to represent the starting and ending of the schedule, respectively. Table 2 presents the setup costs between the processing of the jobs. For this example, we use $T' = 180$ (batch due date) and $h\% = 50\%$. Figure 1 shows an optimal solution for $P_m|d_{batch}, h\%_{batch}|TC$, where 81%, 63% and 54% of the total processing time of batches 1, 2 and 3 are processed before the batch due date, respectively. The makespan of the schedule is equal to 271, and the total sequence-dependent setup cost is equal to 24.

Solution approaches for the classical non-preemptive parallel machines problem cannot be used to solve $P_m|d_{batch}, h\%_{batch}|TC$. The optimal solution for the classical parallel machine scheduling problem is shown in Figure 2, with the same makespan value as before but lesser setup cost, 21. However, this solution is infeasible to our problem since batch 2 is only processed 47% of the total time before T' . Therefore, efficient heuristics to solve $P_m|d_{batch}, h\%_{batch}|TC$ must be derived.

Table 1: Jobs data

Job id	P_i	Batch
0	0	-
1	99	1
2	109	2
3	69	1
4	63	2
5	111	3
6	87	3
7	0	-

Table 2: Setup costs

c_{ij}	0	1	2	3	4	5	6	7
0	0	2	3	2	5	4	5	0
1	2	0	2	1	1	2	5	2
2	3	2	0	4	3	3	2	3
3	2	2	1	0	2	4	3	2
4	5	3	2	3	0	1	4	5
5	4	3	3	3	5	0	5	4
6	5	3	2	4	5	4	0	5
7	0	2	3	2	5	4	5	0

1.2. Contributions

The contributions of this research are three-fold. First, a new optimization problem is introduced. It arises when managing a health care mobile dental programs and

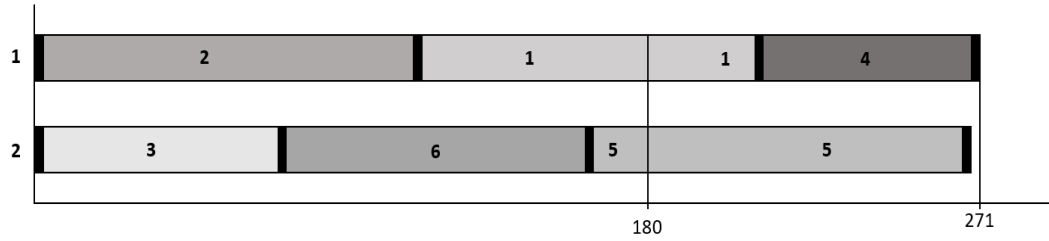


Figure 1: Scheduled obtained considering batch due date.

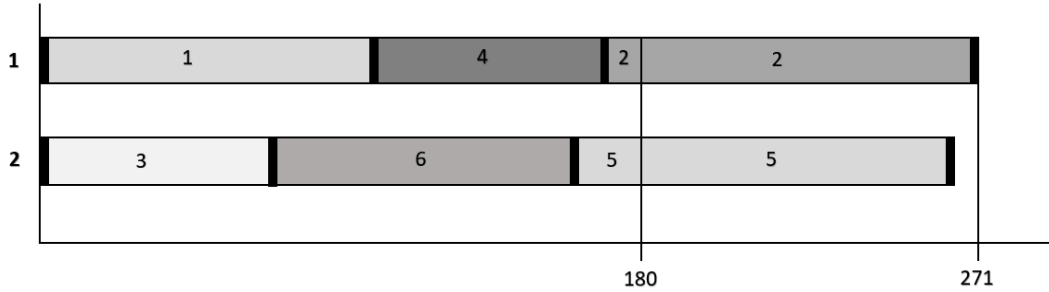


Figure 2: Scheduled obtained without considering batch due date.

differs from the existing scheduling literature by considering batch due date with fairness/equitability considerations. The problem has an implementation-specific feature that forces the decision-maker to visit at least 50% of the total students in each district before the end of the first semester. Second, a mixed-integer linear programming (MILP) formulation for the problem is proposed. Third, and due to the complexity of the problem, an efficient algorithm based on genetic algorithms is proposed and it is compared with the MILP formulation and with an additional algorithm reported in the literature. The results indicates the heuristics is able to obtain near-optimal solutions to problems involving up to 150 jobs.

The remaining of this thesis is organized as follows. In Section 2, we present a brief literature review of realted scheduling problems. In Section 3, a MILP formulation is proposed to capture the scheduling problem, and a methodology based on genetic algorithm is presented from which two variants are obtained for solving large-sized $P_m|d_{batch}, h\%_{batch}| TC$ instances involving up to 150 jobs. Section 4 describes the computational implementation, where the 31 instances and the parameter setting are presented. This section also reports

the results obtained from the algorithm testing along with a discussion of each one of the tested methods. Section 5 closes this research with some concluding remarks and future works proposals.



2. Literature review

The parallel machine scheduling addressed in this paper considers batch due dates and sequence-dependent setup costs. To the author's knowledge, the problem as defined here has not been studied before. However, it is related to some existing problems reported in the literature.

Non-preemptive parallel machines scheduling problem

For the case of the problem addressed in this paper, it is an extension of the identical parallel machines schedule problem where the objective function is the minimization of the total cost, which is compounded by a C_{max} cost and a sequence-dependent setup cost. It has been proved that $P||C_{max}$ is NP-Hard (Garey & Johnson, 1979). Chiaselotti et al. (2010) studied this problem proposes a $n\log(n)$ algorithm that combines partial solutions that are obtained by partitioning the set of jobs into suitable families of subsets. Chang et al. (2004) addressed the same problem but, in this case, machines can process jobs in batches, therefore this involves two decision problems: job batching and batch scheduling. They proposed a simulated annealing approach for it. Biskup et al. (2008) studied a non-preemptive identical parallel machines problem using total tardiness as performance measure. They proposed a new heuristic motivated by the parallel nature of the problem trying to consider the interdependencies of the scheduling and assignment of tasks and compared it with other heuristics and algorithms previously proposed showing computational results. Baptiste et al. (2015) presented a two identical parallel machines, considering, besides, a single operator in order to minimize the makespan. A pseudo-polynomial time algorithm is exhibited to generate an optimal solution within the free changing mode. Chung et al. (2019) addressed the same problem but instead a single operator, they considered molds as resource constraints, it means that two or more jobs with the same mold requirement cannot be processed on the same or different machines at the same time. They proposed three heuristics with a worst-case performance ratio of 3/2. Page & Solis-Oba (2020) studied an unrelated machine problem in order to minimize the makespan. They considered that the set of jobs is partitioned in bags, this implies that no

two jobs belonging to the same bag can be scheduled on the same machine. They presented a simple b -approximation algorithm and a polynomial-time approximation scheme for the case with machine types where both the number of machine types and bags are constant.

Scheduling problems with job batch and batch due dates

Two additional features are encountered in the problem addressed on this paper, job batches and due dates. One investigation that included both characteristics on it is Brucker et al. (1994), where a parallel machine batch scheduling problem with group deadlines (in this case, the batches are compounded by groups of jobs) and sequence-independent setup times was studied, a problem that is NP-hard even considering identical jobs for each batch. In this case, the problem was solved using dynamic programming. Cheng & Kovalyov (2000) is another example that studied these topics, here a batch scheduling problem with due dates on unrelated parallel machines was treated, and it implies three decision problems: scheduling, batching, and due dates assignment. Both previous examples study batching as a decision problem, this paper, instead, includes batch belonging as a job feature that is part of the problem inputs and a common due date just for a fraction of each one of the batches.

Surprisingly, there are only a few articles that consider batch due dates in the scheduling literature. Daganzo (1989) and Peterkofsky & Daganzo (1990) both considered a crane scheduling problem where ships are divided into holds, only one crane can work on a hold at a time and cranes can be moved freely from hold to hold. Thus, both articles considered an open shop problem with identical machines, where ships can be considered as a batch (with a common due date), each hold as a job, and where preemption is allowed. The objective function is to minimize the sum of weighted batch tardiness. In Daganzo (1989) a heuristic procedure was developed and some optimal solutions are founded for special cases, while Peterkofsky & Daganzo (1990) presented a branch and bound method which, for this model, minimizes delay costs. On the other hand, in Yin et al. (2013) a batch delivery single-machine scheduling problem was studied, in which jobs have an assignable common due window. The objective is to find the optimal size and location of the window,

the optimal dispatch date for each job, as well as an optimal job sequence to minimize a cost function based on earliness, tardiness, holding time, window location, window size, and batch delivery. Here, a dynamic programming algorithm was proposed, and it solved the problem, optimally, in $O(n^8)$ time. Su et al. (2013) presented the customer order problem where jobs are scheduled on a set of identical parallel machines and dispatched in batches that have due dates. Minimization of maximum lateness was used as performance measure. Three heuristics, based on Earliest Due Date rule (EDD) were proposed and their tight worst-case bounds are found. Finally, Chung et al. (2014) addressed a canned food scheduling problem, which was treated as an identical parallel machine problem with batch due date to minimize the total tardiness. Two heuristics were proposed, based on the Largest processing time (LPT) rule, to find the near-optimal solution. Unlike previous research, the problem that we present in this paper does not consider normal batch due date, instead it considers a due date just for a fraction of each batch. Both due date and the fraction are common for all batches.

Scheduling problems with sequence dependent setup costs

Another feature that is related to the problem addressed in this thesis is the sequence-dependent setup costs. Many applications consider this feature, Pinedo (2008), for example, described a paper bag factory where setup is needed when the machine switches between types of paper bag and its value depends on the degree of similarity between consecutive batches. Other practical situations arise in the chemical, pharmaceutical, food processing, metal, and paper industries (Srikar & Ghosh, 1986; Bianco et al., 1988; Bitran & Gilbert, 1990; Kim & Bobrowski, 1994).

Even though we are dealing with sequence-dependent setup costs, the two performance measures of setup time and setup cost can be considered as equivalent if setup time and setup cost are proportional, Allahverdi & Soroush (2008).

For the case of parallel machines, Heady & Zhu (1998) addressed an identical parallel machine problem with sequence-dependent setup times, where some machines may not be able to process some jobs. A heuristic was proposed for the sum of earliness and

tardiness costs minimization. Mendes et al. (2002) and Gendreau et al. (2001) addressed the same problem but in this case, the performance measure was makespan minimization. Mendes et al. (2002) proposed two heuristics, one based in tabu search and the other a memetic approach that is a combination of a population-based method with local search procedures. Tahar et al. (2006) studied the same problem including job splitting. They proposed a heuristic based on linear programming modeling and compared their results with a lower bound to test the performance of the method. On the other hand, Montoya-Torres et al. (2009) included release dates instead of job splitting, and they proposed a heuristic algorithm, which uses a strategy of random generation of various execution sequences. Pereira & de Carvalho (2007) proposed a branch and price algorithm for an unrelated parallel machines problem with sequence-dependent setup times and availability dates for the machines and release dates for the jobs to minimize a regular additive cost function. They developed a new column generation accelerating method, termed “primal box”, and a specific branching variable selection rule that significantly reduces the number of explored nodes. Anderson et al. (2013) proposed a network-based MIP formulation model for the sequence-dependent setup time problem with the sum of earliness and tardiness as performance measure. They showed that their MIP model is more efficient than the earlier existing models in terms of computational time for large problems. Yuzukirmizi (2017) studied a parallel machine problem with sequence-dependent setup times and a single server in order to minimize makespan. He proposed a decomposition procedure as a solution approach. This procedure is based on dividing the problem into two decision steps where the outcomes are optimal for their counterpart problems. Silva et al. (2018) addressed an unrelated parallel machines problem with sequence-dependent setup times. The objective considered here is makespan minimization and they proposed several algorithms for it, but a “Fix and Optimize” heuristic with a variable neighborhood search presented the best results. Báez et al. (2019) tackled the same problem, instead, Yepes-Borrero et al. (2020) also considered sequence-dependent resources. Báez et al. (2019) presented a two phases algorithm (construction and improvement) that is

performed employing a general variable neighborhood search, while Yepes-Borrero et al. (2020) proposed three metaheuristics following two approaches: a first approach that ignores the information about additional resources in the constructive phase, and a second approach that takes into account this information about the resources. Bastos & Resendo (2020) addressed an unrelated parallel machines problem with sequence-dependent setup times with resource constraints with total completion time as performance measure. A two-step approach is presented to solve real-size instances. This method uses a relaxation of the model proposed. Afterward, a heuristic algorithm is presented to adjust these solutions.

As previous researches reveal, problems with objective functions that depend on sequence-dependent setup costs have been already studied. However, the problem addressed in this paper considers also a C_{max} cost in its objective function, in addition to the features named in the previous paragraphs. Hence, we can summarize the main differences between the problem addressed in this paper and the ones reported in the literature:

- It considers a partial batch completion deadline, which is common to all batches.
- It works with an objective function based in the total cost of the schedule, which is compounded by a C_{max} cost and a sequence-dependent setup cost of the schedule.
- It consider a planning horizon divided in two periods.
- It assumes that jobs are distributed in batches with fairness/equitability considerations ensuring a proportion of jobs in each batch must be processed during the first period.

Table 3 summarizes the problems of the articles presented in the literature review to understand better the main differences between them and the problem addressed in this paper.

Table 3: Summary of literature review.

Authors	Machines configuration	Due dates(h%)	Setup times or costs	Performance measure
Chiaselotti et al. (2010)	Identical parallel machines	No	No	C_{max}
Chang et al. (2004)	Identical parallel machines	No	No	C_{max}
Biskup et al. (2008)	Identical parallel machines	No	No	Total tardiness
Baptiste et al. (2015)	Two Identical parallel machines	No	No	C_{max}
Chung et al. (2019)	Two Identical parallel machines	No	No	C_{max}
Page & Solis-Oba (2020)	Unrelated parallel machines	No	No	C_{max}
Brucker et al. (1994)	Unrelated parallel machines	Group deadlines (100%)	S.i. setup times	Feasible schedule
Cheng & Kovalyov (2000)	Unrelated parallel machines	Job deadlines (100%)	Constant setup time	Feasible schedule
Yin et al. (2013)	Single machine	Job due window (100%)	No	Total cost function
Su et al. (2013)	Identical parallel machines	Batch due dates (100%)	No	Maximum lateness
Chung et al. (2014)	Identical parallel machines	Batch due dates (100%)	No	Total tardiness
Heady & Zhu (1998)	Identical parallel machines	No	S.d. setup times	Sum of earliness and tardiness cost
Mendes et al. (2002)	Identical parallel machines	No	S.d. setup times	C_{max}
Tahar et al. (2006)	Identical parallel machines	No	S.d. setup times	C_{max}
Montoya-Torres et al. (2009)	Identical parallel machines	No	S.d. setup times	C_{max}
Pereira & de Carvalho (2007)	Unrelated parallel machines	Job due dates (100%)	S.d. setup times	Total weighted tardiness
Anderson et al. (2013)	Identical parallel machines	Job due dates (100%)	S.d. setup times	Sum of earliness and tardiness
Yuzukirmizi (2017)	Identical parallel machines	No	S.d. setup times	C_{max}
Silva et al. (2018)	Unrelated parallel machines	No	S.d. setup times	C_{max}
Bález et al. (2019)	Unrelated parallel machines	No	S.d. setup times	C_{max}
Yepes-Borrero et al. (2020)	Unrelated parallel machines	No	S.d. setup times	C_{max}
Bastos & Resendo (2020)	Unrelated parallel machines	No	S.d. setup times	Total completion time
Our work	Identical parallel machines	Common batch due date (50%)	S.d. setup costs	Sum of setup and C_{max} costs

S.d.: Sequence dependent
S.i.: Sequence independent



3. Methodology

The method to solve a scheduling problem will depend of its feature, like machine configuration, performance measure or job preemption, for example. A problem like $P_m | d_{batch}, h\%_{batch} | TC$ can be solved in many ways, but due to it is a new problem as it was exposed in the above sections, many methods from the literature are not able to solve it or they must to be adapted. Thus, an iterative approach based in genetic algorithms is proposed to solve the problem addressed in this research. This approach is an adapted version of the algorithm proposed in Prins (2004).

This section is structured as follows: First, the complexity of the problem is determined through a comparison with other problems from the literature. Then, a mixed-integer linear programming model is proposed to solve the problem based in the scheduling theory. Finally, the genetic algorithm framework is presented, explaining how it operates and each one of its stages.



3.1. Problem complexity

As we set above, scheduling problems and VRP are related, in this sense if we consider a simpler version of the problem, for example, $G = 1$, the problem is similar to a makespan minimization problem, where Guinet (1993) suggests it is equivalent to the VRP with service times. Even more, if we focus on the single machine version of the problem, considering $G = 1$ again, it is equivalent to the Traveling salesman problem with a cluster completion constraint. In conclusion, the problem is NP-hard even for the single machine case.

3.2. Mathematical formulation

In this subsection, we propose a mixed-integer linear programming (MILP) formulation to capture the $P_m | d_{batch}, h\%_{batch} | TC$.

Parameters

P_i : Processing time of job i .

T' : Common batch due date.

h : Portion of each batch that must be processed before T' .

G : Cost incurred by each machine while the processing is running (including setup time)

c_{ij} : Cost of processing job j just after job i .

$$w_{ik} : \begin{cases} 1 & \text{If job } i \text{ belongs to batch } k \\ 0 & \text{otherwise} \end{cases}$$

s : Sequence independent setup time (service time)

Sets

N : Set of jobs, $N = \{0, 1, 2, 3, \dots, n + 1\}$. n indicates the number of jobs, while 0 and $n + 1$ represent fictitious starting and ending jobs, respectively.

M : Set of machines, $M = \{1, 2, 3, \dots, m\}$. m denotes the number of machines.

T : set of time periods, $T = \{1, 2\}$. 1 represents the period before T' , while 2 denotes the period after T' .

B : Set of batches, $B = \{1, 2, 3, \dots, b\}$. b denotes the total number of batches.

Decision variables

x_{il}^t : Time units of job i that are processed in machine l during the period t .

C_{max} : makespan.

$$z_{ijl} : \begin{cases} 1 & \text{If job } i \text{ is processed just before job } j \text{ in machine } l \\ 0 & \text{otherwise} \end{cases}$$

$$q_{il}^t : \begin{cases} 1 & \text{If job } i \text{ is processed in machine } l \text{ during the period } t \\ 0 & \text{otherwise} \end{cases}$$

u_i : Latent variable of node i to avoid generating subtours.

Formulation

$$\text{Minimize } \sum_{i=0}^{n+1} \sum_{j=0, i \neq j}^{n+1} \sum_{l=1}^m c_{ij} z_{ijl} + mGC_{max} \quad (1)$$

$$\sum_{t=1}^{|T|} \sum_{l=1}^m x_{il}^t = P_i, \quad \forall i = 1, \dots, n \quad (2)$$

$$\sum_{i=1}^n x_{il}^1 + s \sum_{i=0}^{n+1} q_{il}^1 \leq T', \quad \forall l \in M \quad (3)$$

$$\sum_{i=1}^n w_{ik} \sum_{l=1}^m x_{il}^1 \geq \sum_{i=1}^n w_{ik} P_i h, \quad \forall k \in B \quad (4)$$

$$x_{il}^t \leq P_i q_{il}^t, \quad \forall l \in M, t \in T, \forall i = 1, \dots, n \quad (5)$$

$$q_{0,l}^1 = 1, \quad \forall l \in M \quad (6)$$

$$q_{n+1,l}^2 = 1, \quad \forall l \in M \quad (7)$$

$$\sum_{i=1}^n \sum_{l=1}^m \sum_{t=1}^{|T|} q_{il}^t \leq n + m \quad (8)$$

$$q_{il}^{t_1} + q_{ir}^{t_2} \leq 1, \quad \forall i \in N, t_1 \in T, t_2 \in T, l \in M, r \in M, r \neq l \quad (9)$$

$$z_{ijl} + q_{il}^2 + q_{jl}^1 \leq 2, \quad \forall l \in M, i \in N, j \in N, i \neq j \quad (10)$$

$$\sum_{i=1}^{n+1} z_{0il} = 1, \quad \forall l \in M \quad (11)$$

$$\sum_{i=0}^n z_{in+1l} = 1, \quad \forall l \in M \quad (12)$$

$$\sum_{i=0}^{n+1} \sum_{l=1}^m z_{ijl} = 1, \quad \forall j = 1, \dots, n \quad (13)$$

$$\sum_{i=0}^{n+1} z_{ijl} = \sum_{i=0}^{n+1} z_{jil}, \quad \forall l \in M, \forall j = 1, \dots, n \quad (14)$$

$$u_i - u_j + n z_{ijl} + (n - 2) z_{jil} \leq n - 1 \quad \forall l \in M, \forall i = 0, \dots, n, \forall j = 1, \dots, n + 1, i \neq j \quad (15)$$

$$u_0 = 0 \quad (16)$$

$$z_{ijl} \leq \frac{\sum_{t=1}^{|T|} x_{il}^t}{2P_i} + \frac{\sum_{t=1}^{|T|} x_{jl}^t}{2P_j}, \quad \forall l \in M, i \in N, j \in N, i \neq j \quad (17)$$

$$C_{max} \geq s + \sum_{j=0}^{n+1} \sum_{i=1}^n z_{ijl} (P_i + s), \quad \forall l \in M \quad (18)$$

$$x_{il}^t \in \mathbb{Z}^+ \quad \forall i \in N, l \in M, t \in T \quad (19)$$

$$q_{il}^t \in \{0, 1\} \quad \forall i \in N, l \in M, t \in T \quad (20)$$

$$z_{ijl} \in \{0, 1\} \quad \forall i \in N, j \in N, l \in M \quad (21)$$

$$C_{max} \in \mathbb{Z}^+ \quad (22)$$

$$u_i \in \mathbb{Z}^+ \quad \forall i \in N \quad (23)$$

The objective function (1) minimizes the sequence-dependent and makespan related costs. Constraints (2) ensure that each job i is processed during P_i units of time on a machine during the first and second periods. Constraints (3) limit, for each machine, the total available processing time in first period, which is less or equal than T' (including jobs processing and setup times). Constraints (4) guarantee the processing time of each batch during the first period for at least $h\%$ of its total processing time. Constraints (5) relate x_{il}^t and q_{il}^t , i.e., the former variable can be greater than zero only when q_{il}^t are equal to 1. Constraints (6) and (7) assign fictitious jobs 0 and $n+1$ to each machine in period 1 and 2, respectively. Constraint (8) enforces that at most $n+m$ q_{il}^t -variables take value equal one. n results from the total number of jobs to be scheduled, while m indicates that at most one additional job per machine can be assigned. For each machine, it is possible that a job i can be processed at the end of the period $t = 1$ and continue its processing during period $t = 2$, in this case, the job i can be allocated on the same machine in both periods. Constraints (9) indicate that a job i can assigned to at most one machine in any period, i.e., variable q_{il}^t can not take a value equal to one for a given job in two different machines, independent of the period. Constraints (10) indicate that if a job i is scheduled in the period $t = 2$ on machine l , then a job j , that is processed in the period $t = 1$ ($j \neq i$) cannot be allocated right after i . Constraints (11)-(17) capture the job's position on the machines in order to compute the set-up costs. Constraints (11) and (12) ensure that 0 and $n+1$ are the starting and ending jobs on each machine, respectively. Constraints (13) ensure that each job is assigned to a unique position on a machine. Constraints (14) indicate that if a job i is assigned to a machine l , it has a job scheduled before and after it. Constraints (15) prohibit cycles in the solution and are known as subtour elimination constraints (SECs). These constraints are derived from the Miller-Tucker-Zemlin constraints, and the improvement proposed by ? is applied. Constraints (16) initialize the position variable as 0 in each machine. Constraints (17) indicate that job j can be placed right after job i on machine l only if both jobs are processed in the same machine. Constraints (18) compute C_{max} among the machines. Finally, Constraints (19)-(23) define the domains for all decision variables.

3.3. A genetic algorithm for the $Pm|d_{batch}, h\%_{batch}|TC$

According to Whitley (1994), genetic algorithms (GAs) are a family of computational models inspired by evolution. These algorithms encode a potential solution to a specific problem on a simple chromosome-like data structure and apply crossover and mutation operators (that usually are recombination operators) to these structures in such a way as to preserve critical information.

GAs have been applied to different parallel machines scheduling problems. In Dayong Hu & Zhenqiang Yao (2010) two GAs were proposed for the parallel machine scheduling problem with sequence-dependent setup times. The first of them uses a scheme of random assignment, while the second employs a scheme of greedy assignment, being the latter the one that showed better performance. Adan et al. (2018) studied a real-life problem in a complex manufacturing environment that is characterized by a large product and batch size variety, numerous parallel machines with large capacity differences, sequence and machine-dependent setup times, and machine eligibility constraints. They developed a hybrid GA which is characterized by a local search enhanced crossover mechanism, two additional fast local search procedures, and a user-controlled multi-objective fitness function.

Several researchers have developed a GA as a solution method to related optimization problems. Prins (2004) proposed a hybrid GA for the vehicle routing problem where the number of vehicles is a decision variable. One of its most remarkable features is the optimal splitting procedure used for the interpretation of the solution represented by a chromosome. In Cattanuzza et al. (2014) a memetic algorithm was developed for the multi-trip VRP using some of the procedures of Prins (2004), specifically, its splitting procedure and local search (LS) as mutation operator which is speeded up using a granular search. Vidal et al. (2012) addressed three problems, the multi-depot VRP, the periodic VRP, and the multi-depot periodic VRP with heterogeneous capacitated vehicles and constrained route duration. They developed a metaheuristic that combines the exploration breadth of population-based evolutionary search, the aggressive-improvement capabilities of neighborhood-based metaheuristics, and an advanced population-diversity management scheme. Shuai et al. (2019) developed a multi-objective genetic algorithm to solve the multiple traveling salesman problem. This GA has a scheme of random assignment and uses a combined HGA crossover operator.

A general outline of the GA that we propose can be seen in Algorithm 1. This GA is an adapted version of the algorithm proposed in Prins (2004), where the main differences lie in the heuristic to split the jobs between the machines, in some parameters used to enhance the algorithm performance, like stopping criteria; and in the way that some procedures are applied.

This framework work as follows: initialization (Line 1), and a loop of reproduction and formation of new generations of chromosomes with probability of mutation (Lines 3-7). This loop continues while stopping criteria are not met and it contains a population restart procedure (Lines 9-10).

Algorithm 1: GA outline

```

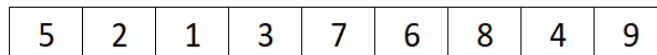
1 Initialize population
2 while Stopping criteria are not met do
3   Parents selection ( $P_1, P_2$ )
4   Crossover and child selection ( $C$ )
5   if  $random(0, 1) \leq p_m$  then
6     | Mutate  $C$  into  $C_m$ 
7   Insert  $C$  (or  $C_m$ ) in population
8   Update stopping criteria variables
9   if restart criteria are met then
10  | restart population

```

3.3.1. Solution representation and evaluation

Chromosomes

A chromosome is a sequence (permutation) S of n jobs, without machine delimiters. It can be interpreted as the order in which a machine must process all jobs, if just one machine performs all jobs one by one. In this sense, S has to be transformed in a feasible solution for $P_m | d_{batch}, h\%_{batch} | TC$ using a splitting technique. The fitness value, $F(S)$, i.e., the total cost function of S , will depend of the way in that S is splitted. Fig. 3 shows a sample of a chromosome of an instance of nine jobs numerated from 1 to 9, however it can be a sample chromosome of different instances with different number of machines or batch distribution. These parameters (together with others) will determine its cost function and its feasibility. A similar approach is proposed in Prins (2004) wherein a splitting algorithm for a vehicle routing problem, where the optimum number of vehicles and the way the jobs are divided are determined generating an auxiliary graph that represents the splitting options. In this case, we have a fixed number of machines and we just have to decide the machine delimiters.



5	2	1	3	7	6	8	4	9
---	---	---	---	---	---	---	---	---

Figure 3: Chromosome sample

Splitting Heuristic

We propose a heuristic to split the chromosomes in order to obtain a solution for $P_m | d_{batch}, h\%_{batch} | TC$ and its respective cost function. This heuristic assumes an equal processing time on each machine, denoted by T^* :

$$T^* = \frac{s(n + m) + \sum_{i \in N} P_i}{m} \quad (24)$$

Where n , m , and s are the number of jobs, number of machines, and the sequence independent setup time. T^* is a theoretical makespan computed as the the total scheduling time are distributed equally among the machines.

The heuristic works as follows. The jobs are assigned in each machine following the sequence given by the chromosome until the first job does not fit in the machine, then the machine is closed and the assignment continues in the next machine. The assignment of a job i to a machine k is given by its completion time and T^* . If the completion time of job i is less than T^* , the job i is assigned to machine k . If the completion time of job i exceed T^* , but it is processed by at least 50% (including processing time and setup time s) at time T^* , then the job i is assigned to machine k . Otherwise, machine k is closed; then, the next machine is opened and job i is assigned to it. Thus, the heuristic works as a Next-Fit heuristic, and it has the same complexity, that is $O(n)$.

Most of the time, the heuristic returns a solution with m machines. However, sometimes it returns a solution with $m + 1$ machines. In this case, the jobs that were assigned to machine $m + 1$ are reassigned to machine m to obtain a viable solution.

The heuristic procedure is explained in detail in Algorithm 2. Lines 1-3 initialize the variable of time (t), the first machine $k = 1$ is opened, and the parameter T^* is calculated, respectively. Lines 4-11 contain the loop that assigns the jobs in the order that they are sorted in the chromosome. Line 5 updates t and Line 6 checks if the Fit rule is met. If not, in Lines 7-9 machine k is closed, machine $k + 1$ is opened, the current job i is assigned to it and t is updated. Otherwise, job i is scheduled on machine k . Finally, Lines 12-14 check if an additional machine was opened and an unfeasible solution was created, then it is corrected as it was explained above.

Fig. 4 illustrates the splitting heuristic (it works with the chromosome sample of Fig. 3). In this sample, we need to split the nine jobs between three machines. The scheduling starts on machine 1, assigning jobs 5, 2, 1 and 3; however, job 3 does not met the fit rule criteria because it exceeds T^* in more than the 50% of the processing time P_3 plus s (assuming that the figure is representative regarding the processing times). Following the heuristic, machine 1 is closed

Algorithm 2: Splitting Heuristic

```

1  $t = s$ 
2  $k = 1$ 
3  $T^* = \frac{s(n+m) + \sum_{i \in N} P_i}{m}$ 
4 for  $i$  in chromosome do
5    $t = t + P_i + s$ 
6   if  $(t - T^*) \geq ((P_i + s)/2)$  then
7      $k = k + 1$ 
8     Assign job  $i$  to machine  $k$ 
9      $t = P_i + 2s$ 
10  else
11    Assign job  $i$  to machine  $k$ 
12 if  $k = M + 1$  then
13   Move jobs from machine  $M+1$  to machine  $M$ 
14   Remove machine  $M+1$ 

```

and job 3 is assigned to machine 2 that now is open. The heuristic continues until obtain the scheduling of the right side of the Fig. 4.

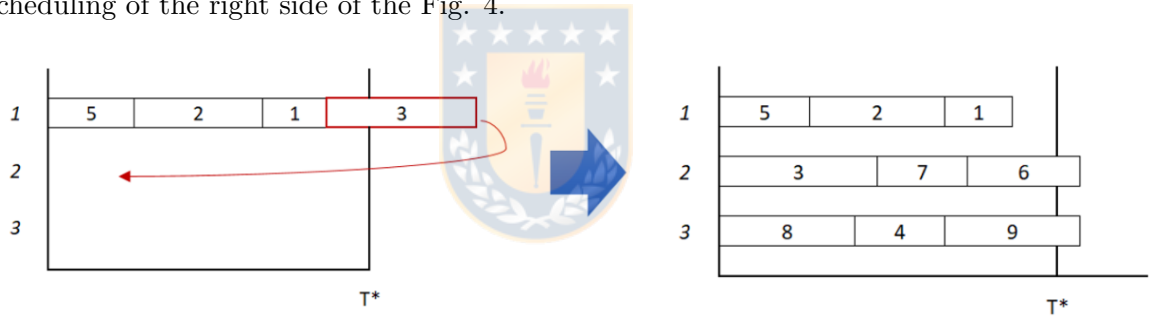


Figure 4: Splitting procedure sample.

3.3.2. Crossover

As the encoding used does not have delimiters, the OX-operator can be used. This operator is suited for cyclic permutations like TSP. Since one parallel machine scheduling solution can give different chromosomes depending on the concatenation order of its jobs, there is no special reason to distinguish a “first” or “last” job. This is why we selected OX.

Fig. 5 shows how OX-operator works. First, two cutting positions i and j are randomly selected (In the example $i = 3$ and $j = 5$). Then, the section of the Parent 1 between i and j is copied into Child 1 maintaining the same positions. Finally, Parent 2 is swept circularly from $j + 1$ onward inserting in Child 1 with the missing jobs. By inverting the roles between Parent 1 and Parent 2, we obtain the second child.

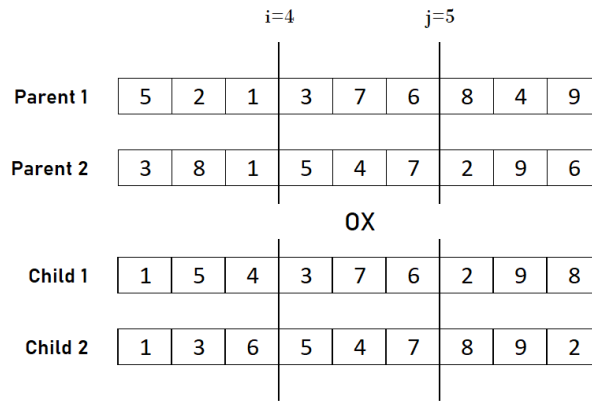


Figure 5: Example of OX crossover

Parent 1 and Parent 2 are selected with the classic binary tournament method: two chromosomes are randomly drawn from the population and the one with the lower cost function is selected. The procedure is repeated twice, to obtain both parents. The child that has to be inserted in the population is randomly selected between children.

3.3.3. Mutation operator

After crossover, the selected child (C) is evaluated using the splitting heuristic and improved applying a local search procedure (LS) with a probability p_m . First, the chromosome is converted using the *Splitting heuristic* to identify the machine delimiters.

Each iteration of LS scans all possible pairs of distinct jobs (u, v). These jobs may belong to the same machine or to different machines. For each pair, the following moves are tested. x and y are the successors of u and v in their respective machines (that could represent the end-start of a machine as well). $R(u)$ indicates the machine that process job u .

- M1. If u is a job, remove u then insert it after v .
- M2. If u and x are jobs, remove them then insert (u, x) after v .
- M3. If u and x are jobs, remove them then insert (x, u) after v .
- M4. If u and v are jobs, swap u and v .
- M5. If u, x and v are jobs, swap (u, x) and v .
- M6. If (u, x) and (v, y) are jobs, swap (u, x) and (v, y) .
- M7. If $R(u) = R(v)$, replace (u, x) and (v, y) by (u, v) and (x, y) .
- M8. If $R(u) \neq R(v)$, replace (u, x) and (v, y) by (u, v) and (x, y) .
- M9. If $R(u) \neq R(v)$, replace (u, x) and (v, y) by (u, y) and (x, v) .

Moves M1–M3 correspond to insertion moves, moves M4–M6 to swaps, move M7 is the well

known $2 - opt$ while M8,M9 extend $2 - opt$ to different machines. Moves like M1 can empty a machine but can also assign new jobs to it later. This is why empty machines are removed only at the end of LS. The final mutated chromosome C_M is converted in a MDCSP solution using the *Splitting heuristic* obtaining a solution with a cost function value $F(C_M) \leq F(C)$.

Figure 6 and Figure 7 show a sample of movement M5 and M8, respectively, to better understand how the movements work. In both cases $u = 2$ and $v = 8$. If these movements generate a improvement in the cost function, they will be applied.

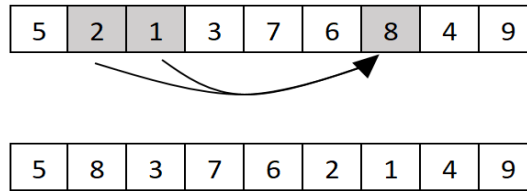


Figure 6: M_5 move sample

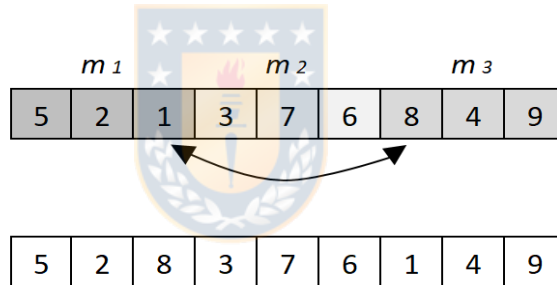


Figure 7: M_8 move sample


At the beginning of the LS each type of movement M_i has the status *active*. At each iteration the LS procedure randomly selects a move among the active moves. The selected move M_i is evaluated and the first improvement is adopted (each time a chromosome is evaluated, the *Split Heuristic* is applied). If the move fails, i.e., the current solution is a local optima in the neighborhood defined by M_i , M_i becomes *inactive* and cannot be selected anymore until another move succeeds. The LS terminates when all the moves are inactive, i.e., a local optima in the neighborhood defined by M1–M9 is reached. As said before, LS scans all possible pairs of distinct job.

Algorithm 3 describe the mutation operator in detail. Here, *mutations* represents a vector with the movement index in random order, and *combinations* represents a vector with all the

possible combinations of jobs (u, v) in random order. C and C_M represent a child and its mutated version, respectively. On the other hand, j references the number of inactive movements.

The local search process can be time-consuming if each movement is performed for each combination of pair of jobs. So, to speed it up, we propose two alternatives. The first is a stopping condition consisting in a maximum number of local search applied to a each chromosome, which is denoted by δ . The second strategy is a granular search, that was proposed by Toth & Vigo (2003) and implemented in Cattaruzza et al. (2014), and it limits the LS only to the $n_{closest}$ (nr , where n is the number of jobs of the instance, and $r \in [0, 1]$ is a granularity threshold restricting the search to nearby vertices) jobs of the job u , where these closest jobs are determined by the sequence-dependent setup costs between them. In this sense, we define two versions of the GA described in this paper. The first one, named as GA_δ , has a limit of δ movements for each chromosome that enters to the mutation process. The second one, named GA_∞ , has no limit for the movements in the mutation process, but the granular search is applied. Except for this difference, the overall scheme of the algorithm is the same.

Algorithm 3: Mutation procedure



```

Input:  $C$ 
Output:  $C, C_M$ 
1 if  $random(0, 1) \leq p_m$  then
2    $i=0$ 
3    $j=0$ 
4    $C_M = C$ 
5   Create combinations vector
6   Create mutations vector
7   while  $j \neq 9$  do
8     for  $(u, v) \in combinations$  do
9       Try movement  $mutations[i]$  on  $(u, v)$  to  $C_M$ 
10      if  $C_M$  improves then
11        Apply  $mutation[i]$  to  $C_M$ 
12        Create  $mutations/\{mutations[i]\}$  vector
13        Create combinations vector
14         $i = -1$ 
15         $j = 0$ 
16        break for
17      else
18         $next(u, v)$ 
19       $j = j + 1$ 
20       $i = i + 1$ 

```

3.3.4. Population structure and initialization

The population is structured as an array Π of σ chromosomes sorted in increasing value of cost. So, the best chromosome with the best objective value is Π_0 .

Clones are not allowed in the population. Following the recommendation exposed in Prins (2004), clones are chromosomes with the same fitness value (instead of equal chromosomes or chromosomes that represent the same solution). This condition works as follows: the cost of any two solutions must be spaced at least by a constant $\Delta > 0$. If all the chromosomes in the population meet this condition, the population is said to be well spaced. Regarding infeasible solutions, these are allowed in the population but a penalty is applied to their cost function. This penalty increases the cost function in a value directly proportional to G , m , and the units of time remaining to be processed to meet the batch completion constraints, t^- . We denote this constant of proportionality as the penalty factor μ . Thus, the fitness value of a chromosome y ($F(y)$) is given by the equation (25), where the term in bold is the total penalty for the solution that chromosome y represents. These rules will ensure a better dispersal of solutions to avoid local minimums and to diminish the risk of premature convergence. In this sense, it is possible that the chromosome Π_0 , that has the best objective value, may not be a feasible solution, so the best solution is given by $\Pi'_0 \in \Pi'$, where $\Pi' \subset \Pi$ contains all the chromosomes in Π that represent feasible solutions sorted in increasing value of cost.

$$F(y) = mGC_{max} + \mu Gmt^- + \sum_{i \in N} \sum_{j \in N} c_{ij} z_{ij}^t \quad (25)$$

The initial population is generated randomly, adding chromosomes one at a time, where each one of the chromosomes must meet the distance condition to obtain a well-spaced population. Later on, in each GA iteration only attempts to replace one chromosome. Therefore, the spacing must be checked in each of them.

3.3.5. Iteration description

Parents and child selection

Parents are selected by binary tournament. Two chromosome are randomly selected from the population and the least-cost one (i.e. the chromosome with better position in the population) becomes the first parent P_1 . The second parent P_2 is selected in the same way. With the crossover two children C_1 and C_2 are generated. To avoid solution degeneration, one child C is selected randomly for replacing a mediocre chromosome Π_k , with $k \geq \lfloor \sigma/2 \rfloor$.

Child mutation and insertion

Once the child is selected, it could be mutated or not with a probability of occurrence p_m . If the child experiments the LS mutation process, the resulting chromosome C_m is attempted to be incorporated into the population replacing the mediocre chromosome randomly selected, as explained above. If $\{\Pi / \{\Pi_k\} \cup \{C_m\}$ is well spaced, the replacement is accepted and Π is re-sorted moving C_m to its respective position. If C_m can not be added into the population, we try to insert the original child C , following the same procedure. If the replacement is accepted, independent of the chromosome, one productive or successful iteration is considered.

An excessive rejection rate or quantity of unproductive iterations appears if Δ or population size σ are too large. A high mutation rate p_m can worsens this situation generating a very compact population in terms of cost. As exposed in Prins (2004), for the VRP a reasonable rejection rate (at most 10% of iterations) is obtained with $\sigma \leq 50$, $0.2 \leq \Delta \leq 5$ and $p_m \leq 0.3$.

3.3.6. Main phase and restarts

The global phase of the algorithm has four stopping criteria: maximum number of productive iterations (α_{max}), maximum number of productive iterations without improving the best solution (β_{max}), when the optimum cost is reached (if it is known) and when the rejection rate reaches an specific level (which can means a too compact population in terms of cost function).

The main phase is followed by a few restarts based on a partial replacement procedure proposed by Cheung et al. (2001) in a genetic algorithm for a facility location problem. They proposed a reshuffling if there is no further improvement in the fitness value or no new replacement in the current population after a considerable number of generations. This procedure is very helpful in obtaining a better solution especially when the population size is small.

This procedure works as follows. Let ρ the number of chromosomes to be replaced (in this case, $\rho = \lfloor \sigma/4 \rfloor$). To replace this portion of chromosomes random chromosomes are created one by one, if a random chromosome Ω is better than the worst chromosome in Π (i.e. $F(\Omega) \leq F(\Pi_\sigma)$) and $\Pi / \{\Pi_\sigma\} \cup \{\Omega\}$ is well spaced, Ω is inserted in the population and one replacement occurs. If not, Ω is crossed with each one of the chromosomes in Π and the best child C is chosen. If C meets the two above conditions, it is inserted instead Ω and one replacement occurs. This process is repeated until reaching ρ replacements.

4. Computational results

4.1. Benchmark instances

We tested the two GAs in three sets of instances. The first set is a real instance that represents a mobile health program in the Ñuble Region in Southern Chile. This instance contains 27 schools divided into 15 districts that have to be attended by 3 MDCs. The cost for the operation of a MDC (i.e. the value of G) is 60.000 Chilean pesos per day. The setup cost has a sequence-dependent cost, that depends of the distance between the schools, plus a fixed cost of 130.000 Chilean pesos, that represents the pickup truck request.

The second set contains fictitious instances created from the real instance, keeping the same magnitudes in their values. In this case, the parameters T' and T are fixed, so other parameters like the number of machines (or MDCs) and batch number are adapted to ensure the instance feasibility.

Finally, the third set corresponds to $R_m | s_{ij} | C_{max}$ instances used in the literature proposed by Vallada & Ruiz (2011). In this case, the instances are adapted for the problem addressed in this paper and the jobs are uniformly assigned to a specific number of batches that depends of the jobs quantity. T' and T are large enough to ensure the feasibility of the instance.

Table 4 describes, in a general way the instances set 1 and 2. Row 3 indicates the instance id, row 4 represents the number of jobs of each instance, while row 5 and row 6 represent the number of machines and the number of batches, respectively. Table 5 follows the same structure but with the instances set 3.

Table 4: General description of instances set 1 and 2.

	Set										
	1					2					
Instance id	1	2	3	4	5	6	7	8	9	10	11
n	27	10	10	15	15	20	20	25	25	30	30
m	3	2	2	2	2	3	3	3	3	4	4
b	15	4	4	6	6	8	8	10	10	12	12

4.2. Implementation

The GAs are coded in Python 3.7.3, compiled with Windows 10 and run on an Intel Core i5 1.6 GHz processor. To test the algorithms the instances described in subsection 5.1 are used,

Table 5: General description of instances set 3

		Set																			
		3																			
Instance id		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
n		10	10	10	10	12	12	12	12	50	50	50	50	100	100	100	100	150	150	150	150
m		2	2	4	4	2	2	4	4	10	10	15	15	10	10	15	15	10	10	15	15
b		2	4	2	4	3	6	3	6	5	10	5	10	5	10	5	10	5	10	5	10

where for each instance, five runs are executed during a maximum time of 3600 seconds if it is necessary.

In order to compare results, the mathematical model described in section 3 is coded and solved with Gurobi 9.0. The model has a time limit to find solutions of 3600 seconds.

4.3. An additional comparison

In addition to the algorithms described above, an adapted version of the GA proposed in Shuai et al. (2019) (referenced as NGSA in this paper) is coded too to compare the performance of them. To adapt the algorithm to the problem addressed in this paper some modifications are applied to the algorithm:

- A population restart process is added, where a percentage of the chromosomes with the worst objective values are replaced.
- Unfeasible chromosomes are allowed (using the same penalization system applied to the proposed GAs).
- Crossover procedure is adapted to the objective of the problem of this thesis.
- Maximum running time of 3600 seconds.

The main parameters values of the algorithm are specified in table 6, which are set according to Shuai et al. (2019), except for the penalty factor and the number of chromosomes to be replaced on the restart procedure.

4.4. Parameters setting

Prins (2004) is used as guide to set the most of the parameters in the GA. To ensure the completion of the main stage and the activation of some restarts during one hour of algorithm running, α_{max} was set in 15000. The rest of the stopping criteria, both those of the main stage and those of the restarts, were set keeping the same proportion that they have in Prins (2004) respect to this latter parameter.

Table 6: Parameters setting for NSGA

Parameter	Value
Population size	100
Mutation probability	0.05
Maximum number of iterations in main phase	1800
Maximum number of iterations in restart phases	360
Chromosomes to replace on restart	30%
Penalty factor	1

On the other hand, three additional parameters were used in the algorithm configuration, two of them, the penalty factor and the movements limit for the mutation process in GA_δ , were set arbitrarily. The first, a value large enough that ensures feasible solutions in the population, and the second, a low enough value that allows a significant acceleration in the whole process of the algorithm. Finally, the third parameter, the granularity threshold was set as in Vidal et al. (2012) where it is used for a genetic algorithm proposed for three variants of the VRP.

Table 7 summarizes the parameters setting in the GA, while Table 8 specifies those parameters that change their values in restart phases. Table 7 is organized as follows, column 1 represent the parameter abbreviation, column 2 shows the name of the parameter, column 3 corresponds to the value for each one of them. Finally, column 4 contains the reference used to set the value.

Table 7: Parameters setting for the GA

Parameter	Name	Value	Reference
σ	Population size	30	Prins (2004)
Δ	Distance between chromosomes	0.5	Prins (2004)
p_{m1}	Mutation probability in main phase	0.05	Prins (2004)
r	Granularity threshold	0.4	Cattaruzza et al. (2014)
μ	Penalty factor	1	-
δ	Movements limit for GA_δ	1000	-
α_{max}	Maximum number of productive iterations	15000	-
β_{max}	Maximum number of productive iterations without improvement	5000	Prins (2004)

Table 8: Parameters setting in restart phases for GA

Parameter	Name	Value
α_{max}	Maximum number of productive iterations	1000
β_{max}	Maximum number of productive iterations without improvement	1000
p_{m2}	Mutation probability in restart phases	0.1

4.5. Experiments on Algorithms

The performance of each algorithm is based on its capacity to find good solutions during the established time, without considering in which moment they were found, where the quality of the solution is determined first, by the GAP due to the nature of the real problem however, if two or more algorithms report the same best GAP, the best solution is determined by its execution time. Table 9 reports the solution for the three algorithms in 5 of the 31 instances to do a preliminary comparison between them, where for each one of them the best, the worst, and the mean result are shown. Both for the MILP model and the algorithms results the GAP (first row) and the running time in seconds (second row) are reported for each instance, where the GAP is calculated respective to the best bound given by the MILP model. The best result in each instance is indicated in bold. Table 10 and Table 11 contains the results in the 31 instances following the same structure of Table 9. One column is added to both tables that represents the Relative Standard Deviation between the results of each instance reported by GA_{∞} .

It is easy to see in Table 9 that the GA_{∞} reports the best results in all the instances, both in best reported result column and mean results column. On the other hand, GA_{δ} reports a good performance too, with a similar behavior to that of GA_{∞} as the complexity of the instances increases, but reporting bigger GAP values. Regarding NSGA, it reports good results in small size instances but it gets noticeably worse in larger instances, even in the instance 10 it found feasible solutions just in one of the five runs.

Table 9: Preliminary testing of algorithms

Instance id	MILP model	GA_{δ}			GA_{∞}			NSGA			Best
		Best	Worst	Mean	Best	Worst	Mean	Best	Worst	Mean	
1	1.316% 3600	0.105% 683.3	0.149% 666	0.123% 609.2	0.076% 1465.6	0.087% *	0.081% 2736.5	0.785% 139.8	5.242% 137.1	2.148% 134.2	0.076% 1465.6
2	0% 19.22	0% 0.44	0% 2.12	0% 0.92	0% 0.19	0% 1.65	0% 0.90	0% 0.22	0% 1.78	0% 0.72	0% 0.19
4	0% 441.67	0% 40.2	0% 53.0	0% 47.9	0% 17.7	0% 62.2	0% 34.0	0.013% 41.7	0.099% 40.2	0.054% 41.2	0% 17.7
6	0.048% 3600	0.061% 264.8	0.090% 168.6	0.076% 233.5	0.047% 773.6	0.049% 390.9	0.048% 808.1	0.093% 62.7	0.643% 61.5	0.255% 62.5	0.047% 773.6
10	× 3600	0.15% 659.0	0.29% 417.8	0.218% 494.1	0.14% 3600	0.15% 3600	0.148% 3481.7	2.356% 235.3	2.356% 235.3	2.356% 235.3	0.14% 3600

× : No solutions found.

Table 10: Results on set 1 and 2

Instance id	MILP model	GA_∞			Relative Std. Dev.
		Best	Worst	Mean	
1	1.316%	0.069%	0.09%	0.08%	0.0039%
	3600	1014.5	394.3	1168.7	
2	0%	0%	0%	0%	0%
	19.22	0.55	2.12	0.94	
3	0%	0%	0%	0%	0%
	5.27	1.03	8.17	5.82	
4	0%	0%	0%	0%	0%
	441.7	40.2	53.01	47.86	
5	0.0122%	0.012%	0.013%	0.013%	0.0004%
	3600	253.42	531.2	343.73	
6	0.048%	0.061%	0.09%	0.076%	0.0009%
	3600	264.78	168.57	233.46	
7	0.0114%	0.008%	0.008%	0.008%	0%
	3600	405.14	864.45	706.4	
8	×	0.13%	0.165%	0.15%	0.0153%
	3600	3600	167.36	1185.41	
9	0.755%	0.103%	0.124%	0.193%	0.0076%
	3600	1909.49	3600	2194.84	
10	×	0.146%	0.287%	0.217%	0.0063%
	3600	658.98	417.84	494.1	
11	7.33%	0.173%	0.234%	0.211%	0.0258%
	3600	3368.12	3600	3506.45	

× : No solutions found.

4.6. Effect of the movements in the local search

In some instances, like instance 6 and instance 10, GA_δ reports significantly lower execution times than those reported by GA_∞ . If we observe the plots of Fig. 8 and Fig. 9, where the axis y represents the number of successful insertions and axis x represents the number of iterations, it is possible to appreciate that the algorithm stops prematurely because it reaches the rejection rate limit (that we set in 25%). This may be that, due to the size of these instances is not too large, the algorithm begins to have a more compact population and probably falls in a local optimum (that can be interpreted as the break point of the plots of Figure 8 and Figure 9), from which is hard to get out because of the LS movements limit.

Regarding GA_∞ , its iterations become too long in terms of time. This can be appreciated in Fig. 10 that shows a plot of successful insertions vs. number of iterations in instance 21 in 3600 seconds of algorithm running. Here we can see that the algorithm managed to do about 1200 iterations during that time, instead of GA_δ that managed to do about 13000 iterations during the same running time as it is possible to appreciate in Fig. 11. This increase in the average

Table 11: Results on set 3

Instance id	MILP model	GA_{∞}			Relative Std. Dev.
		Best	Worst	Mean	
12	0%	0%	0%	0%	0%
	2.0	0.11	1.55	0.81	
13	0%	0%	0%	0%	0%
	10.81	8.77	25.69	16.69	
14	0%	0%	0%	0%	0%
	259.0	0.93	4.8	2.63	
15	0%	0%	0%	0%	0%
	511.6	1.80	9.54	4.98	
16	0%	0%	0%	0%	0%
	7.52	3.22	13.74	7.56	
17	8.66%	8.56%	8.56%	8.56%	0%
	3600	63.3	727.9	381.6	
18	0%	0%	0%	0%	0%
	3401.4	1.56	26.4	10.6	
19	0%	0%	0%	0%	0%
	1153.14	3.99	113.67	47.21	
20	×	2.82%	3.08%	2.92%	0.1305%
	3600	3600	3600	3600	
21	×	3.58%	4.41%	4.05%	0.3147%
	3600	3600	815.86	3043.17	
22	×	5.88%	6.14%	6.03%	0.0951%
	3600	3600	3600	3600	
23	×	6.0%	6.95%	6.37%	0.3633%
	3600	3600	3600	3600	
24	×	1.59%	1.98%	1.75%	0.1328%
	3600	3600	3600	3600	
25	×	2.01%	2.29%	2.17%	0.0979%
	3600	3600	3600	3600	
26	×	-	-	-	0.1322%
	3600	3600	3600	3600	
27	×	3.13%	3.56%	3.30%	0.1626%
	3600	3600	3600	3600	
28	×	-	-	-	0.0747%
	3600	3600	3600	3600	
29	×	-	-	-	0.419%
	3600	3600	3600	3600	
30	×	-	-	-	0.2643%
	3600	3600	3600	3600	
31	×	-	-	-	0.1214%
	3600	3600	3600	3600	

× : No solutions found, - : Unknown

execution times of the iterations is mainly due to the mutation process, specifically because of the significant increase in the number of combinations of pairs that can be formed when the instance size grows and since it does not have an additional stopping criterion, takes much longer to find the local optimum of the neighborhood defined by each mutation movements. This is supported by the plots exposed in Figure 12 and Figure 13, where is possible to see that when the number of jobs is doubled, the number of iterations in GA_∞ is reduced to a 40%, while in GA_δ it is reduced only to a 60%, approximately. However, with fewer iterations GA_∞ is capable to find better solutions than the other methods.

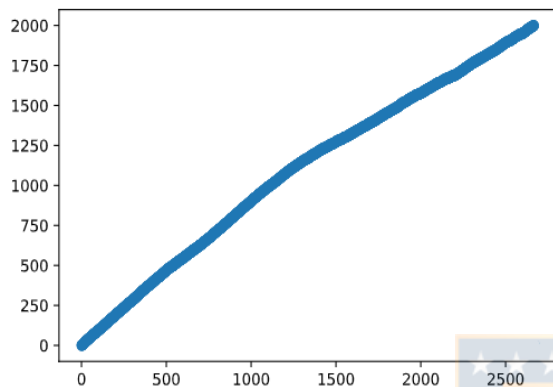


Figure 8: Number of successful insertions vs. Number of iterations in instance 6 of GA_δ

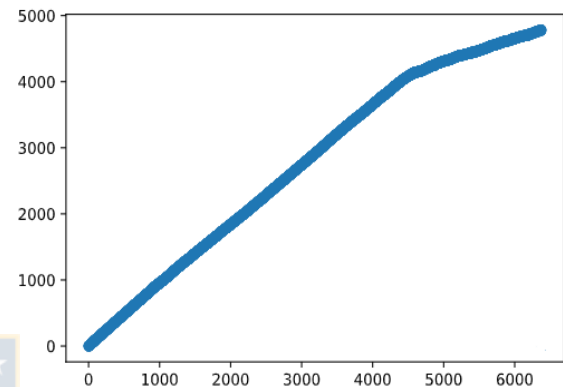


Figure 9: Number of successful insertions vs. Number of iterations in instance 10 of GA_δ

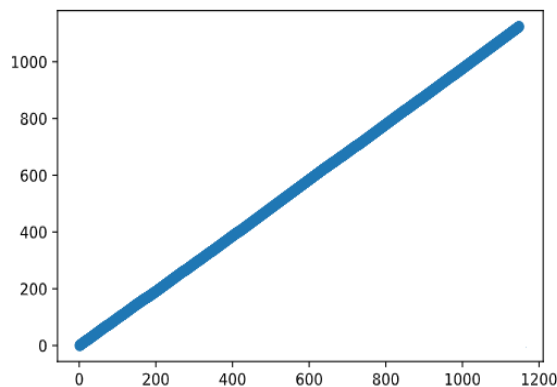


Figure 10: Number of successful insertions vs. Number of iterations in instance 21 of GA_∞

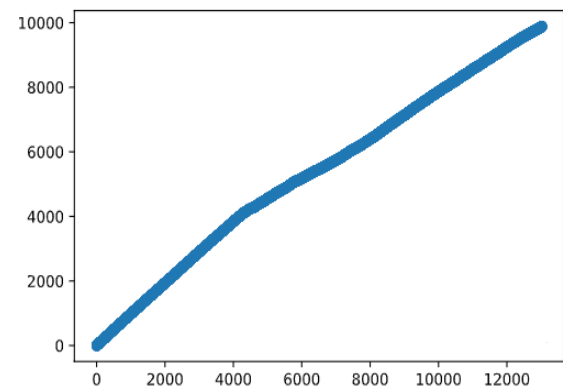


Figure 11: Number of successful insertions vs. Number of iterations in instance 21 of GA_δ

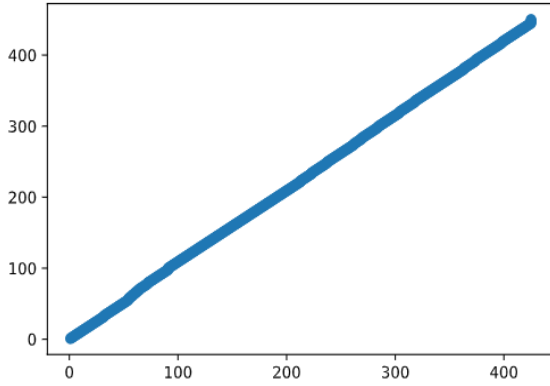


Figure 12: Number of successful insertions vs. Number of iterations in instance 24 of GA_∞

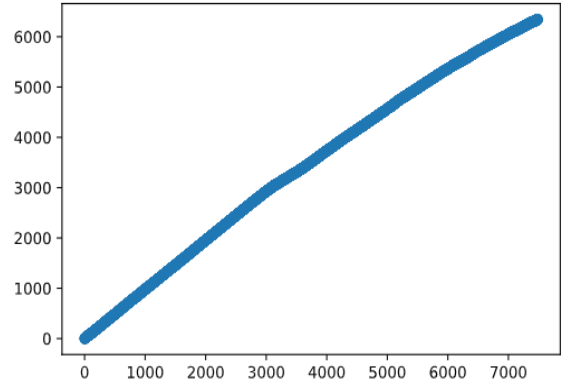


Figure 13: Number of successful insertions vs. Number of iterations in instance 24 of GA_δ

4.7. Comparison with the mathematical model

Section 2.3 demonstrates that $P_m|d_{batch}, h\%_{batch}|TC$ is a new problem in the literature, due to it there are no solutions to compare the proposed methods. In this sense, all the experiments were compared with the exact model that found optimal solutions for 10 instances, feasible solutions for 17 instances, and lower bounds for 26 of them, which is an indicator of the problem complexity, that can be seen more explicitly when we compare the execution time in instance 2 and instance 4, where the difference of job quantity represents an increase of 50%, but the execution time experiences an increase of 2200%, approximately.

If we see the results reported in Table 10, and Table 11 it is possible to appreciate that GA_∞ reaches the optimum in all the instances where it is known, and the algorithm does it in less time than the MILP model. On the other hand, if we focus on the instances where the model can not find the optimum, but it finds feasible solutions, it is easy to see that GA_∞ report better GAP values. Finally, in those instances where the MILP model does not find any solutions, GA_∞ always finds quality and sparsely dispersed solutions.

4.8. Comparing with the literature

Regarding *NSGA*, it was tested in five instances. For small size instances, it shows good results, for example, in instance 2 it found the optimal in all the experiments in very low times. However, its performance noticeably deteriorates as the size of the instances increases, reaching not being able to find feasible solutions in instances of $n = 30$ upwards (For instance 10 it found only one feasible solution). The latter mainly due to two facts: First, its framework has a more random approach compared with GA_∞ and GA_δ ; second, the crossover process of the *NSGA*

modifies the structure of the chromosomes to a lesser extent (compared with the GAs proposed in this research) so it is easier to obtain an unfeasible chromosome from unfeasible parents. Due to the latter $NSGA$ was not tested in the rest of the instances.

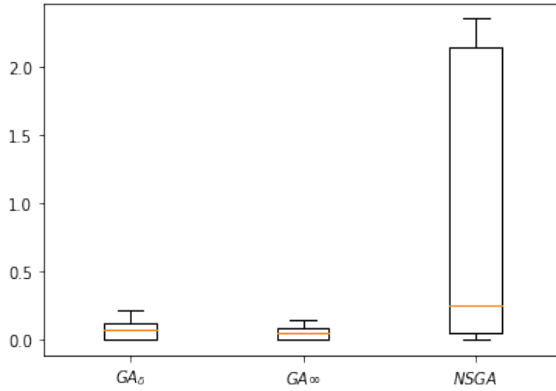


Figure 14: Box plot of the GAP obtained in instances 1-5

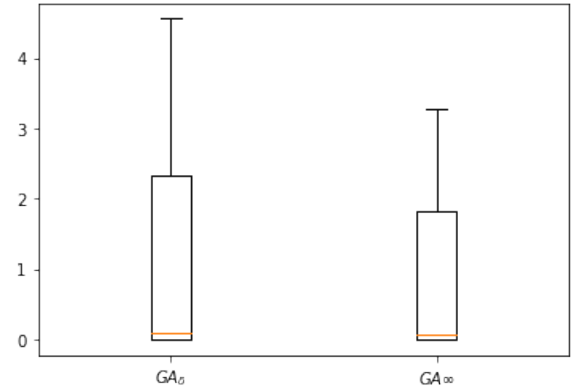


Figure 15: Box plot of the GAP obtained in instances 1-10

Comparing the $NSGA$ performance with GA_{∞} , on average, latter performs better even though $NSGA$ is very competitive in small-size instances. Regarding the results dispersion, GA_{∞} reports clearly less dispersed results, which is supported by Figure 14. In conclusion, GA_{∞} shows a more stable behavior, reporting good GAP values in small, mid and large-size instances, in addition to presenting a low dispersion of them.

4.8.1. Effect of T'

The value of T' is decisive for the effect that the batch due date constraint will have on the solution obtained from a specific instance. It is logical to think that a value of T' large enough will nullify the effect of the constraint and a very small value will make the instance infeasible. To better appreciate this effect we take instance 17 as example.

Figure 16 represents the optimum solution of instance 17 with the original value of T' which is reported in Table 11. Figure 17 and Figure 18 represent the optimum solution of instance 17, but with an increase of 25% and 50% on T' , respectively. All the results reported in these figures are obtained from the MILP model.

It is easy to see that modifications in the value of T' has effects on the solution that involve cost function, running times, and problem complexity, where a later due date can have a positive effect in all of them. For the case of Figure 18, the value of T' is so large that the solution is the same as the one that would be obtained for the $Pm|sc_{ij}|TC$. Regarding the running time, it is possible to see that while the value of T' affects the solution, the complexity of the problem is

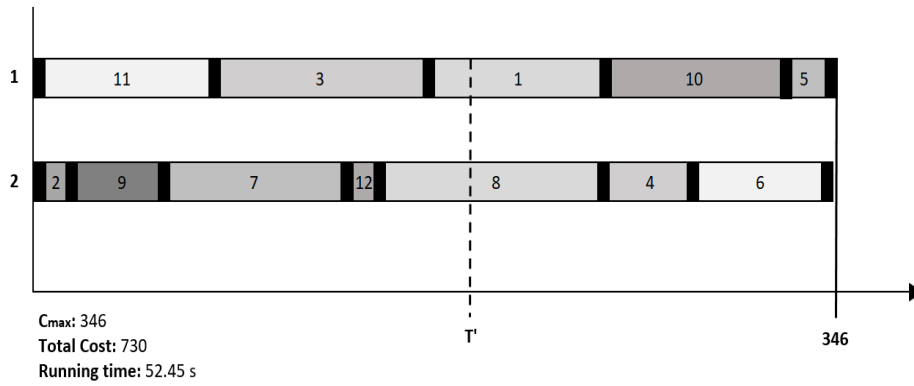


Figure 16: Optimum solution for instance 17 with the original T' value

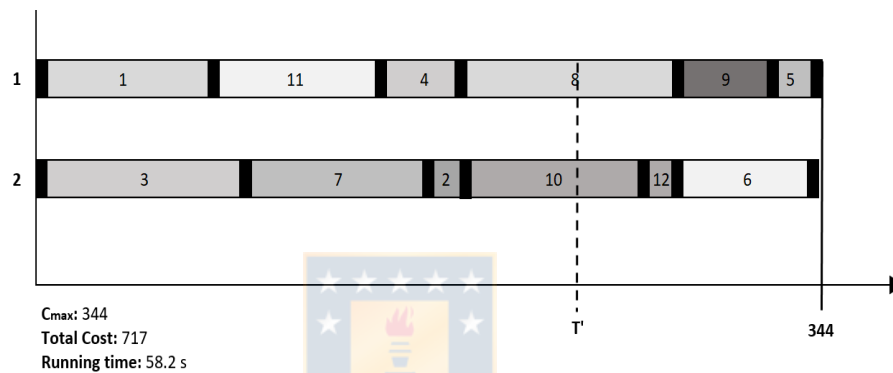


Figure 17: Optimum solution for instance 17 with a 25% of T' increase

greater than that of $Pm|sc_{ij}|TC$, but when the effect of the due date is null, the complexity is practically the same which, based on the running times, is noticeably lower.

These behaviors can be verified with the data from Table 12, which reports the results of the model and the algorithm for the “No due date” case, the original case, and for a 25% of T' increase. For each case the GAP and the running time is reported. Regarding running times of the MILP model, when no due date is considered all the instances report a significant reduction in the running time, except for those like instance 2 where the due date has no effect in the optimal cost function for the original T' value. In some cases, where the model can not find solutions, without due date it can. The algorithm reports a similar behavior.

With this data is clear that T' has an effect on the problem, both in complexity and in the objective values. Specifically, for the instances presented in this paper, the due date can generate an increase in the total cost function of up to 7-8%.

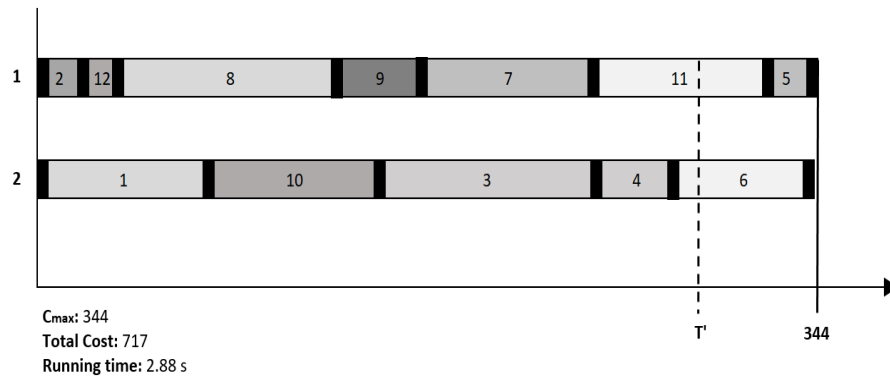


Figure 18: Optimum solution for instance 17 with a 50% of T' increase



Table 12: Model and algorithm sensitivity respect to T'

Instance id	MILP model						GA_{∞}					
	No due date		Original due date		25% due date postponement		No due date		Original due date		25% due date postponement	
	GAP	Time	GAP	Time	GAP	Time	GAP	Time	GAP	Time	GAP	Time
1	0.04%	3600	1.316%	3600	0.09%	3600	0.03%	1490.1	0.081%	2736.5	0.05%	457.7
2	0%	19.8	0%	19.22	0%	14.3	0%	0.58	0%	0.94	0%	0.3
3	0%	4.85	0%	5.27	0%	4.66	0%	0.77	0%	0.622	0%	0.58
4	0%	32.06	0%	441.7	0%	90.73	0%	3.3	0%	47.86	0%	4.2
5	0%	13.97	0.0122%	3600	0%	184.96	0%	0.44	0.013%	343.73	0%	4.9
6	0.03%	3600	0.048%	3600	0.05%	3600	0.037%	101.4	0.076%	233.5	0.047%	1745.4
7	0%	325.22	0.0114%	3600	0%	318.8	0%	18.46	0.008%	706.4	0%	128.8
8	0.064%	3600	×	3600	0.67%	3600	0.055%	258.5	0.15%	1185.4	0.067%	3570
9	0.16%	3600	0.755%	3600	0.76%	3600	0.156%	1598.9	0.193%	2194.8	0.082%	3600
10	0.127%	3600	×	3600	2.21%	3600	0.086%	1904.4	0.217%	494.1	0.098%	3600
11	0.169%	3600	7.33%	3600	0.25%	3600	0.157%	1385.4	0.211%	3506.45	0.163%	268.4
12	0%	1.43	0%	2.0	0%	1.95	0%	0.34	0%	0.812	0%	0.21
13	0%	1.28	0%	10.81	0%	6.8	0%	1.37	0%	16.59	0%	2.76
14	0%	52.66	0%	259.0	0%	149.64	0%	0.44	0%	2.63	0%	1.32
15	0%	176.58	0%	511.65	0%	63.12	0%	0.67	0%	4.98	0%	0.72
8	0%	1.33	0%	7.52	0%	2.96	0%	7.53	0%	7.56	0%	2.91
9	×	3600	×	3600	×	3600	3.58%	3600	4.05%	3043.2	3.64%	3600

×: No solutions found.



5. Conclusions and future works

In this paper we study a new scheduling problem arising in a real situation, named as $P_m|d_{batch}, h\%_{batch}|TC$. A MILP formulation is proposed for the problem addressed, but due to the complexity of the problem (it is NP-Hard), we develop two genetic algorithms (GA_5 and GA_∞) based on the one proposed in Prins (2004). We propose an adaptation of its *Split* procedure considering the problem's conditions (fixed number of machines) to evaluate the chromosomes. The algorithms were tested in 12 instances obtained from a real-life application and instances in the literature. Both proposed *GAs* find optimum solutions when it is known, and report lower running times than the exact model. In most of the instances, GA_∞ performs better and report less dispersed results, despite that the algorithm experiences a considerable increase in the mean execution time of each iteration when the instance size increases.

Besides, to have another point of comparison, we code an adapted version of the GA proposed in Shuai et al. (2019) (NSGA), focusing it according to the addressed problem's objective function. The proposed approach also outperforms NSGA which obtains feasible solutions only to small and mid-size instances.

For future research, we suggest improving the mutation process by adding better stopping criteria and the splitting heuristic for chromosome evaluation. Also, it is necessary to realize the corresponding parameter tuning for the algorithms proposed to boost their performance. On the other hand, this research can open a new promising research direction related to batch conditions in the scheduling theory combining it with features already studied.

References

- Adan, J., Akcay, A., Stokkermans, J., & van den Dobbelaars, R. (2018). A hybrid genetic algorithm for parallel machine scheduling at semiconductor back-end production. In *28th International Conference on Automated Planning and Scheduling, ICAPS 2018* (pp. 298–302). 28th International Conference on Automated Planning and Scheduling, ICAPS 2018 ; Conference date: 24-06-2018 Through 29-06-2018.
- Allahverdi, A., & Soroush, H. (2008). The significance of reducing setup times/setup costs. *European Journal of Operational Research*, *187*, 978–984.
- Anderson, B. E., Blocher, J. D., Bretthauer, K. M., & Venkataramanan, M. A. (2013). An efficient network-based formulation for sequence dependent setup scheduling on parallel identical machines. *Mathematical and Computer Modelling*, *57*, 483 – 493.
- Baptiste, P., Rebaine, D., & Zouba, M. (2015). Solving the two identical parallel machine problem with a single operator. In *2015 International Conference on Industrial Engineering and Systems Management (IESM)* (pp. 502–507).
- Bastos, C. E. N., & Resendo, L. C. (2020). Two-step approach for scheduling jobs to non-related parallel machines with sequence dependent setup times applying job splitting. *Computers Industrial Engineering*, *145*, 106500.
- Bianco, L., Ricciardelli, S., Rinaldi, G., & Sassano, A. (1988). Scheduling tasks with sequence-dependent processing times. *Naval Research Logistics (NRL)*, *35*, 177–184.
- Biskup, D., Herrmann, J., & Gupta, J. N. (2008). Scheduling identical parallel machines to minimize total tardiness. *International Journal of Production Economics*, *115*, 134 – 142.
- Bitran, G., & Gilbert, S. (1990). Sequencing production on parallel machines with two magnitudes of sequence-dependent setup cost. *Manufacturing operations*, *3*, 24–52.
- Brucker, P., Kovalyov, M., & Werner, F. (1994). Parallel machine batch scheduling with deadlines and sequence independent set-ups, .
- Báez, S., Angel-Bello, F., Alvarez, A., & Melián-Batista, B. (2019). A hybrid metaheuristic algorithm for a parallel machine scheduling problem with dependent setup times. *Computers Industrial Engineering*, *131*, 295 – 305.

- Cattaruzza, D., Absi, N., Feillet, D., & Vidal, T. (2014). A memetic algorithm for the multi trip vehicle routing problem. *European Journal of Operational Research*, *236*, 833 – 848. Vehicle Routing and Distribution Logistics.
- Chang, P., Damodaran, P., & Melouk, S. (2004). Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, *42*, 4211–4220.
- Cheng, E. T. C., & Kovalyov, M. Y. (2000). Parallel machine batching and scheduling with deadlines. *Journal of Scheduling*, *3*, 109–123.
- Cheung, B., Langevin, A., & Villeneuve, B. (2001). High performing evolutionary techniques for solving complex location problems in industrial system design. *Journal of Intelligent Manufacturing*, *12*, 455–466.
- Chiaselotti, G., Gualtieri, M. I., & Pietramala, P. (2010). Minimizing the makespan in nonpre-emptive parallel machine scheduling problem. *J. Math. Model. Algorithms*, *9*, 39–51.
- Chung, T., Gupta, J. N., Zhao, H., & Werner, F. (2019). Minimizing the makespan on two identical parallel machines with mold constraints. *Computers Operations Research*, *105*, 141 – 155.
- Chung, T.-P., Liao, C.-J., & Smith, M. (2014). A canned food scheduling problem with batch due date. *Engineering Optimization*, *46*, 1284–1294.
- Daganzo, C. F. (1989). The crane scheduling problem. *Transportation Research Part B: Methodological*, *23*, 159 – 175.
- Dayong Hu, & Zhenqiang Yao (2010). Genetic algorithms for parallel machine scheduling with setup times. In *The 2nd International Conference on Information Science and Engineering* (pp. 1233–1236).
- Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the theory of NP-Completeness..* 4W.H. Freeman and Co: San Francisco.
- Gendreau, M., Laporte, G., & Guimarães, E. M. (2001). A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, *133*, 183 – 189.

- Graham, R., Lawler, E., Lenstra, J., & Kan, A. (1979). *Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey* volume 5 of *Annals of Discrete Mathematics*. Elsevier.
- Guinet, A. (1993). Scheduling sequence-dependent jobs on identical parallel machines to minimize completion time criteria. *International Journal of Production Research*, *31*, 1579–1594.
- Heady, R. B., & Zhu, Z. (1998). Minimizing the sum of job earliness and tardiness in a multi-machine system. *International Journal of Production Research*, *36*, 1619–1632.
- Kim, S. C., & Bobrowski, P. M. (1994). Impact of sequence-dependent setup time on job shop scheduling performance. *International Journal of Production Research*, *32*, 1503–1520.
- Mendes, A. S., Müller, F. M., França, P. M., & Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, *13*, 143–154.
- Montoya-Torres, J. R., Soto-Ferrari, M., Gonzalez-Solano, F., & Alfonso-Lizarazo, E. H. (2009). Machine scheduling with sequence-dependent setup times using a randomized search heuristic. In *2009 International Conference on Computers Industrial Engineering* (pp. 28–33).
- Page, D. R., & Solis-Oba, R. (2020). Makespan minimization on unrelated parallel machines with a few bags. *Theoretical Computer Science*, *821*, 34 – 44.
- Pereira, M. J., & de Carvalho, J. M. V. (2007). A branch-and-price algorithm for scheduling parallel machines with sequence dependent setup times. *European Journal of Operational Research*, *176*, 1508 – 1527.
- Peterkofsky, R. I., & Daganzo, C. F. (1990). A branch and bound solution method for the crane scheduling problem. *Transportation Research Part B: Methodological*, *24*, 159 – 172.
- Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. (3rd ed.). Springer Publishing Company, Incorporated.
- Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers Operations Research*, *31*, 1985 – 2002.
- Shuai, Y., Yunfeng, S., & Kai, Z. (2019). An effective method for solving multiple travelling salesman problem based on nsga-ii. *Systems Science & Control Engineering*, *7*, 108–116.

- Silva, L. G., Ferreira Rego, M., de Assis, L. P., & Vivas Andrade, A. (2018). Algorithms for the unrelated parallel machine scheduling problem with sequence dependent setup times. In *2018 37th International Conference of the Chilean Computer Science Society (SCCC)* (pp. 1–8).
- Srikar, B. N., & Ghosh, S. (1986). A milp model for the n-job, m-stage flowshop with sequence dependent set-up times. *International Journal of Production Research*, *24*, 1459–1474.
- Su, L.-H., Chen, P.-S., & Chen, S.-Y. (2013). Scheduling on parallel machines to minimise maximum lateness for the customer order problem. *International Journal of Systems Science*, *44*, 926–936.
- Tahar, D. N., Yalaoui, F., Chu, C., & Amodeo, L. (2006). A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*, *99*, 63 – 73. Control and Management of Productive Systems.
- Toth, P., & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, *15*, 333–346.
- Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, *211*, 612 – 622.
- Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., & Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, *60*, 611–624.
- Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, *4*, 65–85.
- Yepes-Borrero, J. C., Villa, F., Perea, F., & Caballero-Villalobos, J. P. (2020). Grasp algorithm for the unrelated parallel machine scheduling problem with setup times and additional resources. *Expert Systems with Applications*, *141*, 112959.
- Yin, Y., Cheng, T., Hsu, C.-J., & Wu, C.-C. (2013). Single-machine batch delivery scheduling with an assignable common due window. *Omega*, *41*, 216 – 225. Management science and environmental issues.
- Yuzukirmizi, M. (2017). Scheduling jobs on parallel machines with sequence dependent setup times and a single server, .